**Copy elision** (also known as copy omission) is a compiler optimization method that prevents objects from being duplicated or copied. It makes ‘*returning by value’* or ‘*pass-by-value’* feasible in practice. In simple terms, the compiler prevents the making of extra copies which results in saving space and better the program complexity(both time and space); Hence making the code more optimized. Nowadays, almost every compiler uses it.

This also means fewer objects can be created, so you also can’t rely on a specific number of destructors being called. Or we can conclude that the compiler gets some special power in which they can print according to their utmost feasibility.

#include<iostream>

#include<memory>

#include<string\_view>

#include <array>

//1. Guaranteed Copy/ Move Elision

auto factory(){

return std::make\_unique<int>();

}

// 2. Begining of `consexpr` support in STL

// 3. `constexpr' lambdas

// 4. std::string\_view

// 5. class template argument deduction

// 6. Fold expression

// 7. structured bindings

// 8. if-init expressions

template<typename ...T>

auto add(const T& ... param)

{

return (param + ...);

}

int main()

{

auto widget = factory();

//constexpr auto l = []() {};

//l();

//std::string\_view name = "hello";

std::array<int,4> a = {1,2,3,4};// This is valid for C++14

//std::array data = { 1,2,3,4,5 };// this is valid for C++17

//std::cout << add(1, 2, 3, 4, 5, 5.2); // fold expression call

// structured bindings

/\*std::pair<int, int> values{ 1,2 };

auto [first, second] = values;\*/

//8.

//void func() {

// if (auto [first, second] = values;

// first > 5)

// {

// //do something

// }

//};

return 0;

}